Appendix B1

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US

```
/* amfm_2bit.c file */

/* 2 bits/pixel am/fm halftoning algorithm: part A */
/* Process input image in 4 strips */
/* Assume width of each stripe is multiple of 8 */
/* One row serpentine TDED */
/* One path amfm with dot size error diffusion */
/* A tiff image file containing bit-packed tokens is generated for amfm_pwm.c */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "coef.h"
#include "tiff.h"
#include "allocate.h"

void get_stripe(int,int,unsigned char **,int,int,unsigned char **);
void cut_out_result(int,int,int,int,unsigned char **,unsigned char **);
void amfm(unsigned int,unsigned int,unsigned char *,unsigned char *,short *,\
        short *,unsigned char **,TDEDPARA *,TOKENLUT  *,short *,short *);
void amfm_ed_2bits(unsigned int,unsigned int,unsigned char **,unsigned char **,\
               unsigned char **, TDEDPARA *,TOKENLUT  *,short *,short *);
int main(int argc, char ** argv)
{
  int i,j,width,start_point,cut_offset,cut_width,store_offset,stripe_width;
  FILE * fp;
  struct TIFF_img input_img, output_img, mid;
  time_t first, second;
  unsigned char **stripe, **output_stripe;
  TDEDPARA *tdedpara = &TDEDcoeff[0];
  TOKENLUT  *tokenLUT = &TokenLUT[0] + 30;
  short *dotdensityLUT = &OptDensityLUT[0];
  short *dotsizeLUT = &OptSizeLUT[0];


  if(argc<3) {
```

```
      printf("usage: %s input_img.tif output_img.tif\n",argv[0]);
      return 1;
   }

   /* read the input image  */
   if ((fp=fopen(argv[1], "rb"))==NULL) {
     printf("can not open file %s 1\n",argv[1]);
     exit(2);
   }
   if(read_TIFF(fp,&input_img)) {
     printf("error reading input file\n");
     exit(3);
   }
   fclose(fp);

   if((fp=fopen("dbshalf.tif", "rb"))==NULL) {
     fprintf(stderr, "can not open file: dbshalf.tif\n");
     exit(1);
   }
   if(read_TIFF(fp,&mid))
   {
     fprintf(stderr, "error reading file\n");
     exit(1);
   }
   fclose(fp);

   /* Set variable to do timing of algorithm */
   first = time(NULL);

   /* Modify image width to make sure each strip is multiple of 8 */
   width = floor(input_img.width/32.0)*32;

   /* Allocate memory for entire fm output image. */
   get_TIFF( &output_img, input_img.height, width/4, 'g' );

  /* Process 4 stripes independently */

   cut_width = width/16;

   /* first stripe */
   printf("\nProcess first stripe\n");
   stripe_width = width/4+OVERLAP_WIDTH/2;
   stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width);
   output_stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width/4);
   start_point = 0;

get_stripe(input_img.height,input_img.width,input_img.mono,start_point,stripe_wi
dth,stripe);
   amfm_ed_2bits(input_img.height,stripe_width,stripe,output_stripe,\
              mid.mono,tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
   cut_offset = 0;
   store_offset = 0;
   cut_out_result(input_img.height,cut_offset,cut_width,store_offset,\
             output_stripe,output_img.mono);
   multifree(stripe,2);
```

```
  multifree(output_stripe,2);
  printf("\n");

  /* Second stripe */
  printf("Process second stripe\n");
  stripe_width = width/4+OVERLAP_WIDTH;
  stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width);
  output_stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width/4);
  start_point = width/4-OVERLAP_WIDTH/2;

get_stripe(input_img.height,input_img.width,input_img.mono,start_point,stripe_wi
dth,stripe);
  amfm_ed_2bits(input_img.height,stripe_width,stripe,output_stripe,\
              mid.mono,tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
  cut_offset = OVERLAP_WIDTH/8;
  store_offset = width/16;
  cut_out_result(input_img.height,cut_offset,cut_width,store_offset,\
           output_stripe,output_img.mono);
  multifree(stripe,2);
  multifree(output_stripe,2);
  printf("\n");

  /* Third stripe */
  printf("Process third stripe\n");
  stripe_width = width/4+OVERLAP_WIDTH;
  stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width);
  output_stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width/4);
  start_point = width/2-OVERLAP_WIDTH/2;

get_stripe(input_img.height,input_img.width,input_img.mono,start_point,stripe_wi
dth,stripe);
  amfm_ed_2bits(input_img.height,stripe_width,stripe,output_stripe,\
              mid.mono,tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
  cut_offset = OVERLAP_WIDTH/8;
  store_offset = width/8;
  cut_out_result(input_img.height,cut_offset,cut_width,store_offset,\
           output_stripe,output_img.mono);
  multifree(stripe,2);
  multifree(output_stripe,2);
  printf("\n");

  /* Fourth stripe */
  printf("Process fourth stripe\n");
  stripe_width = width/4+OVERLAP_WIDTH/2;
  stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width);
  output_stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width/4);
  start_point = width/4*3-OVERLAP_WIDTH/2;

get_stripe(input_img.height,input_img.width,input_img.mono,start_point,stripe_wi
dth,stripe);
  amfm_ed_2bits(input_img.height,stripe_width,stripe,output_stripe,\
```

```
                mid.mono,tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
  cut_offset = OVERLAP_WIDTH/8;
  store_offset = width/16*3;
  cut_out_result(input_img.height,cut_offset,cut_width,store_offset,\
                output_stripe,output_img.mono);
  multifree(stripe,2);
  multifree(output_stripe,2);
  printf("\n");


  /* show the run time */
  second = time(NULL);
  fprintf(stdout,"\nFinished AM/FM and writing results.\n");
  fprintf(stdout,"Cum. run time: %f sec.\n",difftime(second,first));

  /* write PWM codes image */
  if( (fp = fopen(argv[2],"wb"))==NULL) {
    printf ("cannot open file %s\n", argv[2]);
    exit(4);
  }

  if(write_TIFF(fp,&output_img)) {
    printf ("\nError writing TIFF file %s\n", argv[2]);
    return 1;
  }
  fclose(fp);

  /* free the space */
  free_TIFF(&(output_img));
  free_TIFF(&(input_img));
  free_TIFF(&(mid));
  fflush(stdout);
  return 0;
}

void amfm_ed_2bits(
  unsigned int height,              /* Input image height */
  unsigned int width,               /* Input image width */
  unsigned char ** contone_img,     /* Input image [height][width] */
  unsigned char ** token_img, /* Output token image [height][width/4] */
  unsigned char ** dbs_screen,      /* DBS screen used in thresholding of fm
part */
  TDEDPARA *tdedpara,         /* Tone-dependent error diffusion parameters */
  TOKENLUT  *tokenLUT,        /* Token LUT used in dot size diffusion */
  short *dotdensityLUT,       /* Optimal dot density curve */
  short *dotsizeLUT)          /* Optimal dot size curve */
{
  short *fm_err,*am_err;
  unsigned int i,j,token_img_width, mod_height;

  /* Initialize first row of fm error buffer */
  srand(1); /* fix the seed */
  fm_err = (short*)malloc(sizeof(short) * (width+2));
  for(j = 0; j<width+2; j++)
    fm_err[j] = (rand()%128-64);  /* Initialization */

  /* initialize first row of am (dot size) error buffer */
```

```
  am_err = (short*)malloc(sizeof(short) * ((width>>1)+4));
  for(i = 0; i<(width>>1)+4; i++)
     am_err[i]=0;        /* Initialization */

  /* Avoid boundary because of pairwise row process */
  if(height & 1)
     mod_height = height - 1;
  else
     mod_height = height;

  /* Process the input image with 2 rows each time */
  for(i=0; i<mod_height; i+=2) {
    if((i%600) == 0) printf("amfm_ed: starting row %d\n", i);
    amfm(width,i,contone_img[i],token_img[i],fm_err,am_err,dbs_screen,\
         tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
  }

  /* Take care the last row if necessary */
  if(height & 1)
  {
    token_img_width = width/4;
    for(j=0;j<token_img_width;j++)
       *(token_img[mod_height]+j) = 0;
  }

  free(fm_err);
  free(am_err);
  return;

}

/* Define macro of FM_EVEN_ROW which does fm for a pair pixels in even row */
#define FM_EVEN_ROW                                                   \
    /* First process FM (dot density) for left pixel in pixel pair. */        \
                                                           \
    /* Get first pixel */                                       \
    pixela = *(img_in_ptr++);                                   \
                                                           \
    /* Use look-up-table to get dot density */                          \
    dotdensity = dotdensityLUT[pixela];                                 \
                                                           \
    /* Compute look-up table entries for tone dependent error diffusion */    \
    tded_ptr = (short*)(tdedpara + dotdensity);                         \
    T2 = tded_ptr[0];                                           \
    DT = tded_ptr[1];                                           \
    W1 = tded_ptr[2];                                           \
    W2 = tded_ptr[3];                                           \
    W3 = tded_ptr[4];                                           \
    W4 = tded_ptr[5];                                           \
                                                                \
    /* compute dotdensity modified by diffused error */                      \
    mod_input = dotdensity + *fm_err_ptr;                       \
                                                           \
    /* Threshold modifed dotdensity */                                  \
    thresholding = mod_input - (dbs_pat_rowptr[j%SCREENWIDTH] * DT + T2);     \
    output = (thresholding > 0) ? 255 : 0;                              \
                                                           \
```

```
    /* Compute weighted errors */                                          \
    error = output - mod_input;                                                  \
    e1 = (W1 * error)>>8;                                                  \
    e2 = (W2 * error)>>8;                                                  \
    e3 = (W3 * error)>>8;                                                  \
    /*e4 = (W4 * error)>>8;*/                                              \
    e4 = error - e1 - e2 -e3;                                              \
    /* Diffuse error forward in 1-D error buffer */                              \
    *(--fm_err_ptr) -= e4;                                                 \
    *(++fm_err_ptr) = fm_tmp - e3;                                         \
    *(++fm_err_ptr) -= e1;                                                 \
    fm_tmp = -e2;                                                    \
                                                                     \
    /* Now process FM (dot density) for right pixel in pixel pair. */             \
    /* Use same TDED parameters as for left pixel. */                            \
                                                                     \
    /* Get second pixel */                                                 \
    pixelb = *(img_in_ptr++);                                              \
                                                                     \
    /* Use look-up-table to get dot density */                                   \
    dotdensity = dotdensityLUT[pixelb];                                          \
                                                                     \
    mod_input = dotdensity + *fm_err_ptr;                                  \
    error = - mod_input;  /* suppress dot firing at this pixel */                \
                                                                     \
    e1 = (W1 * error)>>8;                                                  \
    e2 = (W2 * error)>>8;                                                  \
    e3 = (W3 * error)>>8;                                                  \
    /*e4 = (W4 * error)>>8;*/                                              \
    e4 = error - e1 - e2 -e3;                                              \
    /* Using the tded weights of the left pixel */                                \
    *(--fm_err_ptr) -= e4;                                                 \
    *(++fm_err_ptr) = fm_tmp - e3;                                         \
    *(++fm_err_ptr) -= e1;                                                 \
    fm_tmp = -e2;                                                    \
    j += 2;

/* Define macro of AM_ERROR_EVEN_ROW which computes and distributes
   dot size error for a pair pixels in even row */
#define AM_ERROR_EVEN_ROW                                                  \
    /*      e1 = F1 * error;   */                                          \
    e2 = F2 * error;                                                       \
    e3 = F3 * error;                                                       \
    e4 = (F4 * error);                                                     \
    e1 = error*16 - e2 -e3 -e4;                                            \
    am_err_ptr -= 2;                                                       \
    *(am_err_ptr) -=e4;                                                    \
    *(++am_err_ptr) -=e3;                                                  \
    *(++am_err_ptr) = -e2;                                                 \
    *(++am_err_ptr) -=e1;

/* Define macro of FM_ODD_ROW which does fm for a pair pixels in odd row */
#define FM_ODD_ROW                                                         \
  /* First process FM (dot density) for right pixel in pixel pair */              \
                                                                     \
  /* Get right pixel */                                                    \
    pixela = *(img_in_ptr--);                                              \
```

```
                                                                       \
/* Use look-up-table to get dot density */                                         \
dotdensity = dotdensityLUT[pixela];                                                 \
                                                                       \
/* Compute look-up table entries for tone dependent error diffusion */           \
tded_ptr = (short*)(tdedpara + dotdensity);                                         \
                                                                       \
T2 = tded_ptr[0];                                                            \
DT = tded_ptr[1];                                                            \
W1 = tded_ptr[2];                                                            \
W2 = tded_ptr[3];                                                            \
W3 = tded_ptr[4];                                                            \
W4 = tded_ptr[5];                                                            \
                                                                       \
/* Compute dotdensity modified by diffused error */                                    \
mod_input = dotdensity + *fm_err_ptr;                                        \
                                                                       \
/* suppress this dot and compute the error */                                       \
error = - mod_input;                                                         \
                                                                       \
/* Compute weighted errors */                                                \
e1 = (W1 * error)>>8;                                                        \
e2 = (W2 * error)>>8;                                                        \
e3 = (W3 * error)>>8;                                                        \
/*e4 = ((W4 * error)>>8);*/                                                         \
e4 = error - e1 - e2 -e3;                                                    \
                                                                             \
/* Diffuse error forward in 1-D error buffer */                                     \
*(++fm_err_ptr) -= e4;                                                       \
*(--fm_err_ptr) = fm_tmp - e3;                                               \
*(--fm_err_ptr) -= e1;                                                       \
fm_tmp = -e2;                                                          \
                                                                       \
/* Now process FM (dot density) for Left pixel in a pair */                          \
                                                                             \
/* Get second pixel */                                                       \
pixelb = *(img_in_ptr--);                                                    \
                                                                       \
/* Use look-up-table to get dot density */                                          \
dotdensity = dotdensityLUT[pixelb];                                                 \
                                                                             \
mod_input = dotdensity + *fm_err_ptr;                                        \
                                                                       \
/* Threshold modifed dot density */                                                 \
thresholding = mod_input - (dbs_pat_rowptr[(j-1)%SCREENWIDTH] * DT + T2); \
output = (thresholding > 0) ? 255 : 0;                                              \
                                                                       \
j -= 2;                                                                \
                                                                       \
error = output - mod_input;                                                         \
                                                                             \
e1 = (W1 * error)>>8;                                                        \
e2 = (W2 * error)>>8;                                                        \
e3 = (W3 * error)>>8;                                                        \
/*e4 = (W4 * error)>>8;*/                                                    \
e4 = error - e1 - e2 -e3;                                                    \
                                                                             \
```

```
    *(++fm_err_ptr) -= e4;                                          \
    *(--fm_err_ptr) = fm_tmp - e3;                                  \
    *(--fm_err_ptr) -= e1;                                          \
    fm_tmp = -e2;

/* Define macro of AM_ERROR_ODD_ROW which computes and distributes
   dot size error for a pair pixels in odd row */
#define AM_ERROR_ODD_ROW                                            \
    /*      e1 = F1 * error;  */                                    \
    e2 = F2 * error;                                                \
    e3 = F3 * error;                                                \
    e4 = F4 * error;                                                \
    e1 = error*16 - e2 -e3 -e4;                                     \
                                                              \
    am_err_ptr += 2;                                                \
    *am_err_ptr -= e4;                                              \
    *(--am_err_ptr) -= e3;                                          \
    *(--am_err_ptr) = -e2;                                          \
    *(--am_err_ptr) -= e1;

/* This subroutine only processes 2 rows */
/* Assume width of image is multiple of 8 */
void amfm(
  unsigned int width,   /* Input image width */
  unsigned int i,       /* ith row */
  unsigned char *img_in,      /* ith row of input image array */
  unsigned char *img_out,     /* ith row of output image array */
  short *fm_err,        /* FM error buffer */
  short *am_err,        /* AM error buffer */
  unsigned char ** dbs_screen,      /* dbs_screen[SCREENHEIGHT][SCREENWIDTH] */
  TDEDPARA *tdedpara,   /* Tone-dependent error diffusion parameters */
  TOKENLUT  *tokenLUT,  /* Token look-up-table used in dot size diffusion */
  short *dotdensityLUT, /* Optimal dot density curve */
  short *dotsizeLUT)    /* Optimal dot size curve  */

{
  short fm_tmp,thresholding;
  short *fm_err_ptr, *am_err_ptr;
  short pixela, pixelb, mod_dotsize, output;
  unsigned int j, img_out_width;
  unsigned char bit_pack;
  unsigned char *img_in_ptr, *img_out_ptr, *dbs_pat_rowptr;
  short dotdensity, dotsize, mod_input, error;
  short W1, W2, W3, W4, T2, DT, e1, e2, e3, e4;
  short *tded_ptr, *token_lut_ptr;
  FILE *fp;


  /*-------------------------------------*/
  /* serpentine even rows               */
  /*-------------------------------------*/
      /* Initial points */
  fm_tmp = 0;
  fm_err_ptr = fm_err+1;
  am_err_ptr = am_err+2;
  img_in_ptr = img_in;
  img_out_ptr = img_out;
```

```
/* Get row pointer of dbs pattern */
/* SCREENHEIGHT = 128, module '(i++)%128' can be replace by '(i++)&127' */
dbs_pat_rowptr = dbs_screen[(i++)%SCREENHEIGHT];

/* Index through pixels in pairs */
for(j = 0; j<width;) {

  /* FM halftoning for first pixel pair */
  FM_EVEN_ROW

  /* Begin section on AM halftoning for first pixel pair. */
  /* This section computes 2-bit PWM codes for dot pairs. */
  /* Errors in AM component are diffused using Floyd-Steinberg weights. */
  /* Tokens of left and right pixels along with size error are precomputed */
  /* and stored in tokenLUT */

  /* Get diffused error from dot size error buffer */
  /* This operation can be replace by y=x/16 without affecting */
  /* the quality of the halftone. */
  mod_dotsize = (*am_err_ptr+8)>>4;

  bit_pack = 0;
  if(output) {
    mod_dotsize += dotsizeLUT[pixela];
    /* Get 2 PWM tokens and error corresponding to mod_dotsize */
    /* Then pack the tokens */
    token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
    bit_pack = token_lut_ptr[0]<<6;      /* Get and pack token for left pixel
*/
    bit_pack += token_lut_ptr[1]<<4;     /* Get and pack token for right pixel
*/
    error = token_lut_ptr[2];            /* Get size error for the pixel pair
*/
  }
  else
    error = - mod_dotsize;

  /* Compute and distribute dot size error */
  AM_ERROR_EVEN_ROW

  /* FM halftoning for second pixel pair */
  FM_EVEN_ROW

  /* Begin section on AM halftoning for second pixel pair. */
  /* Same comments for AM halftoning of first pixel pair */
  mod_dotsize = (*am_err_ptr+8)>>4;
  if(output) {
    mod_dotsize += dotsizeLUT[pixela];

    /* Get 2 PWM tokens and error corresponding to mod_dotsize */
    /* Then pack the tokens */
    token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
    bit_pack += token_lut_ptr[0]<<2;     /* Get and pack token for left pixel
*/
    bit_pack += token_lut_ptr[1]; /* Get and pack token for right pixel */
```

```
      error = token_lut_ptr[2];           /* Get size error for the pixel pair
*/
    }
    else
      error = - mod_dotsize;

    /* Compute and distribute dot size error */
    AM_ERROR_EVEN_ROW
    /* write the packed tokens to output image array */
    *(img_out_ptr++) = bit_pack;

  }   /* end of ith row */


/*------------------------------------*/
/* serpentine odd rows                */
/*------------------------------------*/
  img_out_width = width/4;
  fm_tmp = 0;
  /* Set fm error buffer pointer to the end of fm_err buffer */
  fm_err_ptr = fm_err+ width - 1;  /* Offset by 1 */
  /* Set am error buffer pointer to the end of am_err buffer */
  am_err_ptr = am_err + (width>>1);  /* Offset by 2 */
  img_in_ptr = img_in+width*2-2;
  img_out_ptr = img_out+img_out_width*2-1;

  /* Get row pointer of dbs pattern */
  /* SCREENHEIGHT = 128, module 'i%128' can be replace by 'i&127' */
  dbs_pat_rowptr = dbs_screen[i%SCREENHEIGHT];

  /* Index through pixels in pairs */
  bit_pack = 0;
  for(j = width-2; j>2;) {

    /* FM halftoning for first pixel pair */
    FM_ODD_ROW

    /* Begin section on AM halftoning for the first pixel pairl */
    /* This section computes 2-bit PWM codes for dot pairs. */
    /* Errors in AM component are diffused using Floyd-Steinberg weights. */
    /* Tokens of left and right pixels along with size error are precomputed */
    /* and stored in tokenLUT */

    /* Get diffused error from dot size error buffer */
    /* This operation can be replace by y=x/16 without affecting */
    /* the quality of the halftone. */
    mod_dotsize = (*am_err_ptr+8)>>4;
    if(output)
    {
      mod_dotsize += dotsizeLUT[pixelb];
      /* Get 2 PWM tokens and error corresponding to mod_dotsize */
      /* Then pack the tokens */
      token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
      error = token_lut_ptr[2];     /* Get size error for the pixel pair */
      bit_pack += token_lut_ptr[1]<<2;    /* Get and pack token for right pixel
*/
```

```
      bit_pack += token_lut_ptr[0]<<4;     /* Get and pack token for left pixel
*/
    }
    else
      error = - mod_dotsize;
    /* Compute and distribute dot size error */
    AM_ERROR_ODD_ROW

    /* FM halftoning for second pixel pair */
    FM_ODD_ROW

    /* Section on AM halftoning for second pixel pair */
    /* Same comments as on AM halftong for the first pixel pair */
    mod_dotsize = (*am_err_ptr+8)>>4;
    if(output)
    {
      mod_dotsize += dotsizeLUT[pixelb];
      /* Get 2 PWM tokens and error corresponding to mod_dotsize */
      /* Then pack it */
      token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
      error = token_lut_ptr[2];     /* Get size error for the pixel pair */
      bit_pack += token_lut_ptr[1]<<6; /* Get and pack token for right pixel */

      /* Write the packed tokens to the output image array */
      *(img_out_ptr--) = bit_pack;

      bit_pack = token_lut_ptr[0];  /* Get and pack token for left pixel */
    }
    else
    {
      *(img_out_ptr--) = bit_pack;
      bit_pack = 0;
      error = - mod_dotsize;
    }
    /* Compute and distribute dot size error */
    AM_ERROR_ODD_ROW
  }
  /* Take care of the most left three pixels of odd rows */

  /* FM halftoning for the first pixel pair */
  FM_ODD_ROW

  /* AM halftoning for first pixel pair */
  mod_dotsize = (*am_err_ptr+8)>>4;
  if(output)
  {
    mod_dotsize += dotsizeLUT[pixelb];

    /* Get 2 PWM tokens and error corresponding to mod_dotsize */
    /* Then pack the tokens */
    token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
    error = token_lut_ptr[2]; /* Get size error for the pixel pair */
    bit_pack += token_lut_ptr[1]<<2;      /* Get token for right pixel */
    bit_pack += token_lut_ptr[0]<<4;      /* Get token for left pixel */
  }
  else
    error = - mod_dotsize;
```

```
  /* Write the packed tokens to the output image array */
  *(img_out_ptr--) = bit_pack;

  /* Compute and distribute dot size error */
  AM_ERROR_ODD_ROW

  return;
}

void get_stripe(
  int img_height,
  int img_width,
  unsigned char **contone_img,
  int start_point,
  int stripe_width,
  unsigned char **stripe)
{
  int i,j;

  for(i=0;i<img_height;i++)
  for(j=0;j<stripe_width;j++)
      stripe[i][j] = contone_img[i][j+start_point];
}

void cut_out_result(
  int img_height,
  int cut_offset,
  int cut_width,
  int store_offset,
  unsigned char **output_stripe,
  unsigned char **output_img)
{
  int i,j;
  for(i=0;i<img_height;i++)
  for(j=0;j<cut_width;j++)
    output_img[i][j+store_offset]=output_stripe[i][j+cut_offset];
}
```

Appendix B2

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US

```
/* amfm_convert.c file */

/* 2 bits/pixel am/fm halftoning algorithm: part B */
/* Input a tiff file containing 2-bit tokens */
/* Output a tiff file containing pulse width modulation codes */
/* Every pixel is left justified */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#define NEWRIGHT  0xc0
#define NEWLEFT   0x40
#define NEWCENTER 0x00
void token2pwm(int,int,unsigned char **,unsigned char **,unsigned char *);
int main(int argc, char ** argv)
{
  unsigned char pwm[4]={0,21,42,63};
  int i,j;
  FILE * fp;
  struct TIFF_img input_img,output_img;

  if(argc<3) {
      printf("usage: %s token.tif output_img.tif\n",argv[0]);
      return 1;
  }

  printf("Mapping tokens to pulse width modulation codes.\n");

  /* read the input image  */
  if ((fp=fopen(argv[1], "rb"))==NULL) {
    printf("can not open file %s 1\n",argv[1]);
    exit(2);
  }
  if(read_TIFF(fp,&input_img)) {
    printf("error reading input file\n");
    exit(3);
```

```
  }
  fclose(fp);

    /* Allocate memory for entire fm output image. */
  get_TIFF( &output_img, input_img.height, input_img.width*4, 'g' );

  token2pwm(input_img.height, input_img.width, input_img.mono, output_img.mono,
pwm);

  /* write PWM codes image */
  if( (fp = fopen(argv[2],"wb"))==NULL) {
    printf ("cannot open file %s\n", argv[3]);
    exit(5);
  }

  if(write_TIFF(fp,&output_img)) {
    printf ("\nError writing TIFF file %s\n", argv[2]);
    return 1;
  }
  fclose(fp);

  /* free the space */
  free_TIFF(&(input_img));
  free_TIFF(&(output_img));
  fflush(stdout);
  return 0;
}


/* Map tokens to pulse width modulation codes */
void token2pwm(
int img_height,
int img_width,
unsigned char ** token,
unsigned char ** output_pwm,
unsigned char * pwm
)
{
  int i,j,k;

  for(i=0;i<img_height;i++)
      for(j=0;j<img_width;j++)
      {
      output_pwm[i][j*4]    = pwm[(token[i][j]&192)/64] + NEWLEFT;
        output_pwm[i][j*4+1] = pwm[(token[i][j]&48)/16] + NEWLEFT;
        output_pwm[i][j*4+2] = pwm[(token[i][j]&12)/4] + NEWLEFT;
        output_pwm[i][j*4+3] = pwm[token[i][j]&3] + NEWLEFT;
      }
}
```

Appendix B3

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US

```
/* coef.h file */
#define SCREENHEIGHT 128       /* DBS screen height */
#define SCREENWIDTH  128       /* DBS screen width */
#define OVERLAP_WIDTH 16       /* Width of overlapping region */
#define F1 0x0007 /* Floyd-Steinberg Weights 1/16 in Q4 */
#define F2 0x0003 /* Floyd-Steinberg Weights 5/16 in Q4 */
#define F3 0x0005 /* Floyd-Steinberg Weights 3/16 in Q4 */
#define F4 0x0001 /* Floyd-Steinberg Weights 7/16 in Q4 */
typedef struct TDEDPARA
{
short T2;
short DT;
short W1;
short W2;
short W3;
short W4;
} TDEDPARA;
typedef struct TOKENLUT
{
short left;
short right;
short size_error;
} TOKENLUT;
static TDEDPARA TDEDcoeff[256]={
{76, 0, 181, 0, 3, 72},
{76, 0, 181, 0, 3, 72},
{79, 0, 172, 1, 2, 81},
{80, 0, 161, 14, 18, 63},
{82, 0, 159, 1, 37, 59},
{83, 0, 149, 6, 5, 96},
{83, 0, 141, 30, 0, 85},
{85, 0, 138, 13, 0, 105},
{86, 0, 144, 10, 1, 101},
{85, 0, 129, 48, 3, 76},
{86, 0, 123, 31, 1, 101},
{87, 0, 123, 29, 3, 101},
{87, 0, 115, 28, 5, 108},
```

```
{89, 0, 138, 19, 18, 81},
{89, 0, 111, 17, 51, 77},
{88, 0, 115, 31, 0, 110},
{87, 0, 120, 16, 16, 104},
{88, 0, 139, 12, 0, 105},
{89, 0, 122, 19, 17, 98},
{90, 0, 112, 32, 0, 112},
{91, 0, 98, 34, 20, 104},
{90, 10, 123, 16, 26, 91},
{93, 8, 126, 1, 74, 55},
{89, 10, 89, 26, 71, 70},
{89, 10, 89, 22, 43, 102},
{89, 12, 91, 21, 34, 110},
{88, 12, 85, 24, 30, 117},
{88, 14, 85, 23, 30, 118},
{84, 24, 113, 27, 13, 103},
{82, 26, 113, 33, 0, 110},
{83, 26, 109, 29, 9, 109},
{84, 28, 106, 21, 29, 100},
{85, 28, 103, 13, 56, 84},
{96, 2, 102, 16, 57, 81},
{93, 6, 102, 25, 28, 101},
{91, 12, 102, 24, 32, 98},
{96, 2, 103, 24, 23, 106},
{94, 10, 99, 17, 62, 78},
{95, 6, 110, 12, 110, 24},
{97, 4, 114, 12, 112, 18},
{97, 6, 114, 11, 113, 18},
{96, 8, 111, 14, 110, 21},
{94, 12, 102, 17, 109, 28},
{94, 8, 79, 32, 108, 37},
{95, 6, 74, 35, 110, 37},
{97, 2, 70, 35, 111, 40},
{97, 4, 68, 33, 112, 43},
{97, 6, 69, 28, 112, 47},
{98, 6, 70, 22, 114, 50},
{97, 6, 68, 43, 113, 32},
{100, 4, 68, 22, 114, 52},
{99, 6, 71, 24, 112, 49},
{102, 2, 70, 23, 114, 49},
{100, 6, 68, 23, 114, 51},
{100, 8, 66, 22, 116, 52},
{100, 8, 66, 24, 116, 50},
{96, 16, 75, 0, 122, 59},
{95, 16, 63, 0, 127, 66},
{95, 16, 56, 0, 130, 70},
{97, 14, 56, 0, 132, 68},
{97, 16, 59, 0, 132, 65},
{97, 16, 60, 0, 133, 63},
{98, 16, 62, 0, 133, 61},
{95, 26, 98, 0, 109, 49},
{97, 20, 65, 0, 132, 59},
{98, 18, 61, 0, 132, 63},
{99, 18, 63, 0, 131, 62},
{100, 16, 58, 0, 133, 65},
{100, 16, 58, 0, 131, 67},
{101, 16, 60, 0, 131, 65},
```

{101, 16, 63, 0, 129, 64},
{101, 16, 58, 0, 129, 69},
{102, 16, 71, 0, 123, 62},
{103, 8, 68, 23, 114, 51},
{103, 8, 66, 22, 116, 52},
{105, 6, 68, 22, 115, 51},
{106, 4, 70, 22, 114, 50},
{108, 2, 69, 23, 113, 51},
{105, 8, 68, 22, 114, 52},
{108, 6, 70, 20, 115, 51},
{106, 8, 69, 27, 112, 48},
{109, 2, 65, 35, 112, 44},
{110, 4, 69, 34, 111, 42},
{110, 6, 72, 35, 110, 39},
{114, 0, 73, 34, 111, 38},
{110, 12, 94, 21, 108, 33},
{111, 12, 102, 15, 110, 29},
{116, 6, 114, 10, 113, 19},
{96, 16, 92, 16, 67, 81},
{100, 12, 95, 17, 67, 77},
{101, 12, 97, 19, 67, 73},
{99, 4, 101, 20, 45, 90},
{93, 4, 103, 25, 25, 103},
{94, 8, 101, 25, 33, 97},
{78, 24, 99, 26, 19, 112},
{81, 26, 104, 22, 24, 106},
{82, 26, 102, 26, 25, 103},
{91, 26, 109, 14, 46, 87},
{104, 10, 82, 0, 95, 79},
{107, 8, 83, 0, 97, 76},
{105, 8, 87, 2, 84, 83},
{81, 14, 86, 27, 25, 118},
{99, 12, 122, 0, 37, 97},
{102, 10, 117, 0, 45, 94},
{103, 10, 90, 21, 64, 81},
{105, 12, 122, 4, 51, 79},
{101, 12, 126, 9, 29, 92},
{88, 12, 121, 25, 0, 110},
{85, 12, 114, 25, 1, 116},
{89, 10, 109, 23, 10, 114},
{86, 12, 112, 29, 1, 114},
{89, 12, 119, 31, 0, 106},
{94, 10, 123, 37, 1, 95},
{93, 8, 117, 63, 1, 75},
{99, 6, 118, 75, 9, 54},
{97, 6, 120, 43, 3, 90},
{111, 6, 121, 35, 32, 68},
{95, 6, 116, 54, 0, 86},
{107, 6, 125, 39, 15, 77},
{93, 34, 137, 27, 19, 73},
{85, 44, 139, 33, 16, 68},
{87, 48, 146, 31, 23, 56},
{87, 44, 148, 22, 10, 76},
{93, 40, 152, 22, 11, 71},
{97, 44, 159, 4, 28, 65},
{95, 42, 161, 25, 4, 66},
{103, 48, 176, 3, 44, 33},

{101, 56, 165, 27, 55, 9},
{97, 56, 165, 27, 55, 9},
{103, 48, 176, 3, 44, 33},
{117, 42, 161, 25, 4, 66},
{113, 44, 159, 4, 28, 65},
{121, 40, 152, 22, 11, 71},
{123, 44, 148, 22, 10, 76},
{119, 48, 146, 31, 23, 56},
{125, 44, 139, 33, 16, 68},
{127, 34, 137, 27, 19, 73},
{141, 6, 125, 39, 15, 77},
{153, 6, 116, 54, 0, 86},
{137, 6, 121, 35, 32, 68},
{151, 6, 120, 43, 3, 90},
{149, 6, 118, 75, 9, 54},
{153, 8, 117, 63, 1, 75},
{150, 10, 123, 37, 1, 95},
{153, 12, 119, 31, 0, 106},
{156, 12, 112, 29, 1, 114},
{155, 10, 109, 23, 10, 114},
{157, 12, 114, 25, 1, 116},
{154, 12, 121, 25, 0, 110},
{141, 12, 126, 9, 29, 92},
{137, 12, 122, 4, 51, 79},
{141, 10, 90, 21, 64, 81},
{142, 10, 117, 0, 45, 94},
{143, 12, 122, 0, 37, 97},
{159, 14, 86, 27, 25, 118},
{141, 8, 87, 2, 84, 83},
{139, 8, 83, 0, 97, 76},
{140, 10, 82, 0, 95, 79},
{137, 26, 109, 14, 46, 87},
{146, 26, 102, 26, 25, 103},
{147, 26, 104, 22, 24, 106},
{152, 24, 99, 26, 19, 112},
{152, 8, 101, 25, 33, 97},
{157, 4, 103, 25, 25, 103},
{151, 4, 101, 20, 45, 90},
{141, 12, 97, 19, 67, 73},
{142, 12, 95, 17, 67, 77},
{142, 16, 92, 16, 67, 81},
{132, 6, 114, 10, 113, 19},
{131, 12, 102, 15, 110, 29},
{132, 12, 94, 21, 108, 33},
{140, 0, 73, 34, 111, 38},
{138, 6, 72, 35, 110, 39},
{140, 4, 69, 34, 111, 42},
{143, 2, 65, 35, 112, 44},
{140, 8, 69, 27, 112, 48},
{140, 6, 70, 20, 115, 51},
{141, 8, 68, 22, 114, 52},
{144, 2, 69, 23, 113, 51},
{144, 4, 70, 22, 114, 50},
{143, 6, 68, 22, 115, 51},
{143, 8, 66, 22, 116, 52},
{143, 8, 68, 23, 114, 51},
{136, 16, 71, 0, 123, 62},

```
{137, 16, 58, 0, 129, 69},
{137, 16, 63, 0, 129, 64},
{137, 16, 60, 0, 131, 65},
{138, 16, 58, 0, 131, 67},
{138, 16, 58, 0, 133, 65},
{137, 18, 63, 0, 131, 62},
{138, 18, 61, 0, 132, 63},
{137, 20, 65, 0, 132, 59},
{133, 26, 98, 0, 109, 49},
{140, 16, 62, 0, 133, 61},
{141, 16, 60, 0, 133, 63},
{141, 16, 59, 0, 132, 65},
{143, 14, 56, 0, 132, 68},
{143, 16, 56, 0, 130, 70},
{143, 16, 63, 0, 127, 66},
{142, 16, 75, 0, 122, 59},
{146, 8, 66, 24, 116, 50},
{146, 8, 66, 22, 116, 52},
{148, 6, 68, 23, 114, 51},
{150, 2, 70, 23, 114, 49},
{149, 6, 71, 24, 112, 49},
{150, 4, 68, 22, 114, 52},
{151, 6, 68, 43, 113, 32},
{150, 6, 70, 22, 114, 50},
{151, 6, 69, 28, 112, 47},
{153, 4, 68, 33, 112, 43},
{155, 2, 70, 35, 111, 40},
{153, 6, 74, 35, 110, 37},
{152, 8, 79, 32, 108, 37},
{148, 12, 102, 17, 109, 28},
{150, 8, 111, 14, 110, 21},
{151, 6, 114, 11, 113, 18},
{153, 4, 114, 12, 112, 18},
{153, 6, 110, 12, 110, 24},
{150, 10, 99, 17, 62, 78},
{156, 2, 103, 24, 23, 106},
{151, 12, 102, 24, 32, 98},
{155, 6, 102, 25, 28, 101},
{156, 2, 102, 16, 57, 81},
{141, 28, 103, 13, 56, 84},
{142, 28, 106, 21, 29, 100},
{145, 26, 109, 29, 9, 109},
{146, 26, 113, 33, 0, 110},
{146, 24, 113, 27, 13, 103},
{152, 14, 85, 23, 30, 118},
{154, 12, 85, 24, 30, 117},
{153, 12, 91, 21, 34, 110},
{155, 10, 89, 22, 43, 102},
{155, 10, 89, 26, 71, 70},
{153, 8, 126, 1, 74, 55},
{154, 10, 123, 16, 26, 91},
{163, 0, 98, 34, 20, 104},
{164, 0, 112, 32, 0, 112},
{165, 0, 122, 19, 17, 98},
{166, 0, 139, 12, 0, 105},
{167, 0, 120, 16, 16, 104},
{166, 0, 115, 31, 0, 110},
```

```
{165, 0, 111, 17, 51, 77},
{165, 0, 138, 19, 18, 81},
{167, 0, 115, 28, 5, 108},
{167, 0, 123, 29, 3, 101},
{168, 0, 123, 31, 1, 101},
{169, 0, 129, 48, 3, 76},
{168, 0, 144, 10, 1, 101},
{169, 0, 138, 13, 0, 105},
{171, 0, 141, 30, 0, 85},
{171, 0, 149, 6, 5, 96},
{172, 0, 159, 1, 37, 59},
{174, 0, 161, 14, 18, 63},
{175, 0, 172, 1, 2, 81},
{178, 0, 181, 0, 3, 72},
{178, 0, 181, 0, 3, 72},
};
static short OptSizeLUT[256]={
120,
118,
117,
116,
115,
114,
112,
111,
109,
108,
107,
105,
104,
102,
101,
100,
100,
98,
97,
97,
96,
95,
94,
93,
93,
92,
91,
90,
89,
89,
88,
87,
86,
86,
85,
84,
84,
83,
82,
82,
```

81,
80,
80,
79,
79,
78,
78,
78,
77,
77,
77,
76,
76,
76,
75,
75,
75,
75,
74,
74,
74,
73,
73,
72,
72,
72,
71,
71,
71,
70,
70,
70,
70,
70,
70,
70,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,

69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,

69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
68,
68,
68,
68,
67,
67,
67,
67,
66,
66,
65,
65,
65,
64,
64,
63,
63,
62,
62,
61,
61,
60,
60,
59,
59,
58,
58,
57,
57,
56,
56,
55,
55,

```
54,
54,
53,
53,
52,
52,
52,
51,
51,
50,
50,
49,
49,
48,
48,
47,
47,
47,
46,
46,
46,
45,
45,
44,
44,
44,
43,
43,
43,
42,
42,
42,
42,
41,
41,
41,
40,
40,
40,
39,
39,
38,
38,
38,
38,
};
static short OptDensityLUT[256]={
128,
125,
123,
121,
119,
117,
116,
115,
114,
113,
```

113,
112,
112,
111,
111,
111,
111,
111,
110,
110,
110,
110,
110,
110,
109,
109,
109,
109,
109,
109,
109,
108,
108,
108,
108,
108,
108,
108,
108,
107,
107,
107,
107,
107,
107,
107,
107,
107,
107,
107,
107,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
105,
105,
105,

105,
105,
105,
104,
104,
104,
104,
104,
103,
103,
103,
102,
102,
102,
101,
101,
101,
100,
100,
100,
99,
99,
98,
98,
98,
97,
97,
96,
96,
96,
95,
95,
94,
94,
93,
93,
93,
92,
92,
91,
91,
91,
90,
90,
89,
89,
88,
88,
88,
87,
87,
86,
86,
85,
85,
84,
84,

83,
83,
83,
82,
82,
81,
81,
80,
80,
79,
79,
78,
78,
77,
77,
76,
76,
75,
75,
74,
74,
73,
73,
72,
71,
71,
70,
70,
69,
69,
68,
68,
67,
67,
66,
66,
65,
65,
64,
64,
64,
63,
63,
62,
62,
62,
61,
61,
61,
60,
60,
60,
60,
59,
59,
59,
59,

59,
58,
58,
58,
58,
58,
58,
58,
57,
57,
57,
57,
57,
57,
57,
57,
57,
57,
57,
57,
56,
56,
56,
56,
56,
56,
56,
56,
56,
56,
55,
55,
55,
55,
55,
55,
54,
54,
54,
54,
53,
53,
53,
52,
52,
52,
51,
51,
50,
50,
49,
48,
48,
47,
46,
45,
44,

```
43,
42,
41,
40,
39,
37,
36,
35,
33,
31,
29,
27,
24,
20,
16,
11,
7,
0,
};
static TOKENLUT TokenLUT[180]={
{0, 0, 30}, /* -30 */
{0, 0, 29}, /* -29 */
{0, 0, 28}, /* -28 */
{0, 0, 27}, /* -27 */
{0, 0, 26}, /* -26 */
{0, 0, 25}, /* -25 */
{0, 0, 24}, /* -24 */
{0, 0, 23}, /* -23 */
{0, 0, 22}, /* -22 */
{0, 0, 21}, /* -21 */
{0, 0, 20}, /* -20 */
{0, 0, 19}, /* -19 */
{0, 0, 18}, /* -18 */
{0, 0, 17}, /* -17 */
{0, 0, 16}, /* -16 */
{0, 0, 15}, /* -15 */
{0, 0, 14}, /* -14 */
{0, 0, 13}, /* -13 */
{0, 0, 12}, /* -12 */
{0, 0, 11}, /* -11 */
{0, 0, 10}, /* -10 */
{0, 0, 9},  /* -9 */
{0, 0, 8},  /* -8 */
{0, 0, 7},  /* -7 */
{0, 0, 6},  /* -6 */
{0, 0, 5},  /* -5 */
{0, 0, 4},  /* -4 */
{0, 0, 3},  /* -3 */
{0, 0, 2},  /* -2 */
{0, 0, 1},  /* -1 */
{0, 0, 0},  /* 0 */
{0, 0, -1}, /* 1 */
{0, 0, -2}, /* 2 */
{0, 0, -3}, /* 3 */
{0, 0, -4}, /* 4 */
{0, 0, -5}, /* 5 */
{0, 0, -6}, /* 6 */
```

```
{0, 0, -7}, /* 7 */
{0, 0, -8}, /* 8 */
{0, 0, -9}, /* 9 */
{0, 0, -10},        /* 10 */
{0, 0, -11},        /* 11 */
{0, 0, -12},        /* 12 */
{0, 0, -13},        /* 13 */
{0, 0, -14},        /* 14 */
{0, 0, -15},        /* 15 */
{0, 0, -16},        /* 16 */
{0, 0, -17},        /* 17 */
{0, 0, -18},        /* 18 */
{0, 0, -19},        /* 19 */
{0, 0, -20},        /* 20 */
{0, 0, -21},        /* 21 */
{2, 0, 20}, /* 22 */
{2, 0, 19}, /* 23 */
{2, 0, 18}, /* 24 */
{2, 0, 17}, /* 25 */
{2, 0, 16}, /* 26 */
{2, 0, 15}, /* 27 */
{2, 0, 14}, /* 28 */
{2, 0, 13}, /* 29 */
{2, 0, 12}, /* 30 */
{2, 0, 11}, /* 31 */
{2, 0, 10}, /* 32 */
{2, 0, 9},  /* 33 */
{2, 0, 8},  /* 34 */
{2, 0, 7},  /* 35 */
{2, 0, 6},  /* 36 */
{2, 0, 5},  /* 37 */
{2, 0, 4},  /* 38 */
{2, 0, 3},  /* 39 */
{2, 0, 2},  /* 40 */
{2, 0, 1},  /* 41 */
{2, 0, 0},  /* 42 */
{2, 0, -1}, /* 43 */
{2, 0, -2}, /* 44 */
{2, 0, -3}, /* 45 */
{2, 0, -4}, /* 46 */
{2, 0, -5}, /* 47 */
{2, 0, -6}, /* 48 */
{2, 0, -7}, /* 49 */
{2, 0, -8}, /* 50 */
{2, 0, -9}, /* 51 */
{2, 0, -10},        /* 52 */
{3, 0, 10}, /* 53 */
{3, 0, 9},  /* 54 */
{3, 0, 8},  /* 55 */
{3, 0, 7},  /* 56 */
{3, 0, 6},  /* 57 */
{3, 0, 5},  /* 58 */
{3, 0, 4},  /* 59 */
{3, 0, 3},  /* 60 */
{3, 0, 2},  /* 61 */
{3, 0, 1},  /* 62 */
{3, 0, 0},  /* 63 */
```

```
{3, 0, -1}, /* 64 */
{3, 0, -2}, /* 65 */
{3, 0, -3}, /* 66 */
{3, 0, -4}, /* 67 */
{3, 0, -5}, /* 68 */
{3, 0, -6}, /* 69 */
{3, 0, -7}, /* 70 */
{3, 0, -8}, /* 71 */
{3, 0, -9}, /* 72 */
{3, 0, -10},       /* 73 */
{3, 1, 10}, /* 74 */
{3, 1, 9},  /* 75 */
{3, 1, 8},  /* 76 */
{3, 1, 7},  /* 77 */
{3, 1, 6},  /* 78 */
{3, 1, 5},  /* 79 */
{3, 1, 4},  /* 80 */
{3, 1, 3},  /* 81 */
{3, 1, 2},  /* 82 */
{3, 1, 1},  /* 83 */
{3, 1, 0},  /* 84 */
{3, 1, -1}, /* 85 */
{3, 1, -2}, /* 86 */
{3, 1, -3}, /* 87 */
{3, 1, -4}, /* 88 */
{3, 1, -5}, /* 89 */
{3, 1, -6}, /* 90 */
{3, 1, -7}, /* 91 */
{3, 1, -8}, /* 92 */
{3, 1, -9}, /* 93 */
{3, 1, -10},       /* 94 */
{3, 2, 10}, /* 95 */
{3, 2, 9},  /* 96 */
{3, 2, 8},  /* 97 */
{3, 2, 7},  /* 98 */
{3, 2, 6},  /* 99 */
{3, 2, 5},  /* 100 */
{3, 2, 4},  /* 101 */
{3, 2, 3},  /* 102 */
{3, 2, 2},  /* 103 */
{3, 2, 1},  /* 104 */
{3, 2, 0},  /* 105 */
{3, 2, -1}, /* 106 */
{3, 2, -2}, /* 107 */
{3, 2, -3}, /* 108 */
{3, 2, -4}, /* 109 */
{3, 2, -5}, /* 110 */
{3, 2, -6}, /* 111 */
{3, 2, -7}, /* 112 */
{3, 2, -8}, /* 113 */
{3, 2, -9}, /* 114 */
{3, 2, -10},       /* 115 */
{3, 3, 10}, /* 116 */
{3, 3, 9},  /* 117 */
{3, 3, 8},  /* 118 */
{3, 3, 7},  /* 119 */
{3, 3, 6},  /* 120 */
```

```
{3, 3, 5},  /* 121 */
{3, 3, 4},  /* 122 */
{3, 3, 3},  /* 123 */
{3, 3, 2},  /* 124 */
{3, 3, 1},  /* 125 */
{3, 3, 0},  /* 126 */
{3, 3, -1}, /* 127 */
{3, 3, -2}, /* 128 */
{3, 3, -3}, /* 129 */
{3, 3, -4}, /* 130 */
{3, 3, -5}, /* 131 */
{3, 3, -6}, /* 132 */
{3, 3, -7}, /* 133 */
{3, 3, -8}, /* 134 */
{3, 3, -9}, /* 135 */
{3, 3, -10},        /* 136 */
{3, 3, -11},        /* 137 */
{3, 3, -12},        /* 138 */
{3, 3, -13},        /* 139 */
{3, 3, -14},        /* 140 */
{3, 3, -15},        /* 141 */
{3, 3, -16},        /* 142 */
{3, 3, -17},        /* 143 */
{3, 3, -18},        /* 144 */
{3, 3, -19},        /* 145 */
{3, 3, -20},        /* 146 */
{3, 3, -21},        /* 147 */
{3, 3, -22},        /* 148 */
{3, 3, -23},        /* 149 */
};
```